Web Application Development: Java, .Net and Lamp at the Same Time*

Jaime Navón and Pablo Bustos

Computer Science Dept., P.Universidad Católica de Chile
Vicuña Mackenna 4860, Santiago, Chile
{jnavon,pbustos}@ing.puc.cl

Abstract. Web applications are usually built starting from incomplete design documents and proceeding directly to implementation for some specific software platform. The resulting application is usually difficult to change or extend. Although several methodologies have been proposed in the last few years, most of them use a concrete approach that leverages the features of a specific software platform or concrete Web elements. Model-driven development proposals, on the other hand, are difficult to adopt. This paper discusses a successful intermediate approach that allows the designer to work with abstract artifacts that can be readily mapped into any MVC-based (application) framework, independently of which software platform is used. This methodology is simple and easy to learn, even by those who are not platform experts. We present it in terms of a real-life running application for use by local governments in Chile.

1 Introduction

Over the last decade, society has become increasingly dependent on the Internet. In the wake of this phenomenon, software developers are finding themselves under pressure to build more and more complex applications in less and less time. Very often Web applications are built with little or no methodology behind them [1]. It is not unusual to find navigation diagrams mixed together with architectural designs, while important details are not documented at all. The resulting applications exhibit poor quality attributes and are hard to extend or adapt to the ever-changing requirements of the domain.

In the last five years there has been a significant effort in the research community to propose a sound and systematic methodology (method and artifacts). Most of these proposals fall into one of the following two categories:

- A concrete approach that leverages the features of a particular platform (i.e., Java, ASP.NET, etc.) or concrete Web elements (frames, anchors, etc.).
- A relatively abstract approach that is completely independent of the platform and even of the Web itself.

In our experience, neither approach is suitable. The first one is easy to sell: developers see an immediate solution to their concrete problems and it requires significant

* This research is supported in part by the Chilean National Fund for Science and Technology (Fondecyt Project 1020733)

D. Lowe and M. Gaedke (Eds.): ICWE 2005, LNCS 3579, pp. 185–190, 2005.

less effort to learn. In [2] the author defines an extension to UML, called Web Application Extension that models Web elements at a very concrete level. The resulting models are almost ready for implementation, but they remain tightly coupled to the software platform. In [3], the authors propose a methodology for developing all the components typical of Web applications but their methodology and architecture are closely linked to the Java 2 platform Enterprise Edition (J2EE) technology. Although J2EE defines a *de facto* standard for developing multi-tier interactive Web applications, it is not always easy to map these solutions to other popular Web platforms such as PHP (LAMP[1]) or ASP.NET.

As for the second approach, many designers are reluctant to adopt a platform-independent methodology. They find that it has little connection with the world they are dealing with, and feel it will take too long to arrive at a real solution once everything is modeled in the abstract world. This is the case, for example, with the artifacts of some proposals on operational specification of applications. Sequence diagrams, proposed in [4] and [5], cannot specify all possible paths an application would be required to handle. Activity diagrams as proposed in [6] improve the flows specification, but are deficient in components interaction specification. Moreover, most of these proposals share a troubling characteristic: they do not capture the fundamental characteristics of every Web application (i.e., http stateless, and therefore request-response roundtrips and session state). Their level of abstraction goes beyond middleware (i.e., J2EE, ASP.NET) to actually achieve Web independence, and their guides and artifacts can also be used to model a classic non-Web application. This independence, though it may at first seem to be desirable, in fact impedes the identification of essential Web components such as request filters (i.e., logged user session filters). Full platform independence as proposed by the Model Driven Architecture [7] movement is an interesting challenge, but only if the models are specific enough to feed the tools for automatic code generation. Otherwise, completely mapping an abstract model to the implementation platform will be a difficult task.

We have been developing an intermediate approach. It involves a UML-based methodology that is platform independent, but whose artifacts nevertheless can be mapped easily through appropriate restrictions to a specific platform. Furthermore, our approach is general enough to accommodate any Web application whose architecture fits the MVC pattern. In [8] we described the main features of our proposal, which included a first methodology. The key to our approach is a strong coupling with the Web in terms of how applications operate in this context independently of the implementation platforms. In a Web application, or more specifically in the Web tier, the only actor enabled to trigger an event is the user, and the application must always respond with a Web page. The application then operates in round-trip fashion, accepting user requests, processing them, and responding in HTML code. Making use of these restrictions, the level of abstraction of the models can be lowered to achieve an easier implementation, while still maintaining it high enough so the models can be used in any concrete Web platform.

We have tested our approach in two real-life projects. The methodology was easily learned and adopted and the application was built very quickly. Furthermore, though

[1] LAMP is the acronym user for a popular open source platform composed of Linux, Apache, MySql and PHP

using this methodology for the first time the users found the resulting design to be superior in terms of elegance and flexibility. The testing process also yielded important feedback about the methodology's weaknesses, in the light of which we have incorporated significant improvements. In the rest of the paper we present we present our approach trough a running example.

2 Example Application

Our running example is a simple real Web application that allows vehicle owners in Chile to pay vehicle license fees to their local governments over the Internet (CLIP, for Car License Internet Payment). Like most countries, Chile requires that every vehicle have a valid license, which must be renewed on an annual basis. The vehicle owner, known as a "contributor", pays an amount based on the vehicle's appraised and insured values to the local municipality. The contributor must also present a vehicle inspection certificate from an inspection station stating that the vehicle complies with road safety and environmental regulations. The basic requirements are:

- The contributor should be able to pay for her vehicle license over the Internet.
- The contributor should be able to choose between one-time payment, or two semi-annual payments.
- The contributor should be able to select one the insurance company of her choice
- The contributor should give proof of a valid car inspection (certificate number).
- If there are traffic violations associated with the vehicle, the system should allow the contributor to pay any outstanding fines together with the license fee.
- In the case of payments for the second semi-annual period, the system does not check for payment method, insurance, inspection certificates or traffic violations.

Once CLIP has collected all the necessary information for the license payment, it should notify the Payment-Module regarding the transaction to be performed (using Web services) in order to complete the process.

3 Abstract Specification of the Application

Most Web applications are built around the MVC architecture and therefore share a common architectural pattern. The advantages of this pattern have been demonstrated in practice. Some of the most popular implementation frameworks, such as Struts [9], are MVC-based. Consequently, our abstract specification is composed of, a model architecture, a view architecture and a controller architecture. We added a last specification that we call the components interaction specification.

The Model is a conceptual representation of the business domain. This component can be designed using any modern software methodology. As an example, we have successfully used the Unified Software Development Process. In spite of the Model's Web tier independence, specific interfaces for the two main clients, the View and the Controller, must be specified.

The responsibility of the View is to display the Web application's user interface. We have split the View model into two parts: the Interaction model and the Navigation model.

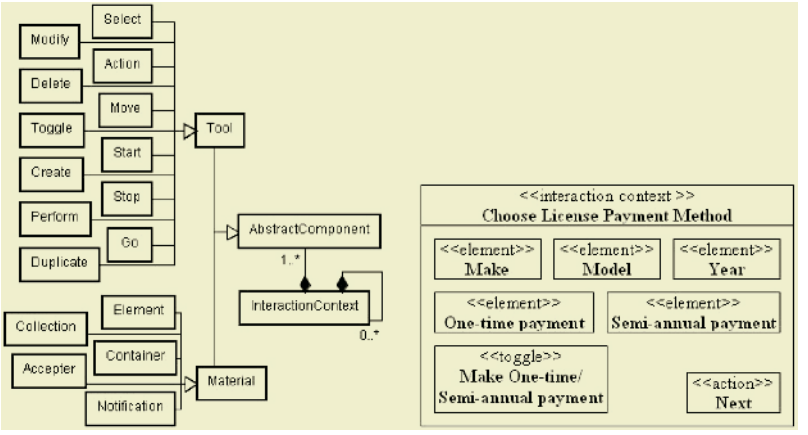

Fig. 1. The interaction context diagram metamodel (left) and one interaction context for the example application (right)

The interaction model of the View (Fig. 1) presents the graphic information of the Web pages displayed to the user. The model is made up of a set of interaction contexts each of which represents the system information and tools displayed to the user at a given moment. The interaction context diagrams are built using a UML extension based on Constantine Canonical Abstract Components ([8], [10]).

The navigation model (Fig. 2) captures the possible flows across the interaction contexts, and it is modeled with UML state charts. These diagrams can describe alternative and conditional flows [13] and the resulting diagrams are relatively small and easy to understand.

The Controller (Fig. 3) ensures coordination of the necessary components in order to process a user request, from reception to response. The Controller behavior is specified through activity UML diagrams. There are three different kinds of activities: model update, interaction context call and filtering.

The last item that must be dealt with in order to complete the description involves the specifics of how the Controller updates the Model and how the View prepares its materials to show the corresponding interaction context. To this end, each activity of the Controller model will map to a corresponding sequence diagram that shows either a Controller-Model interaction or a View-Model interaction.

4 Mapping to Specific Implementation Platforms

It is quite easy to get from here to the actual Java platform or ASP.NET design. We have developed detailed mappings for Java and ASP.NET. These two mappings are based on typical architectures for Java and .Net In a generic MVC Java architecture, a servlet receives the browser requests, filters them using the application filter classes, delegates their processing to the application controller classes, and finally, chooses a JSP to send the display of the interaction context. In the generic ASP.NET architecture, when a request for an ASPX is received, the asp elements are fed by the corresponding CodeBehind. The ASPX then renders the HTML to the browser. When a request from an ASPX form is received, its corresponding Code Behind processes it

by querying and updating the application Model. CodeBehind then transfers control to the next ASPX. If the page is inherited from a filtered Web page, filtering is performed before any page processing.

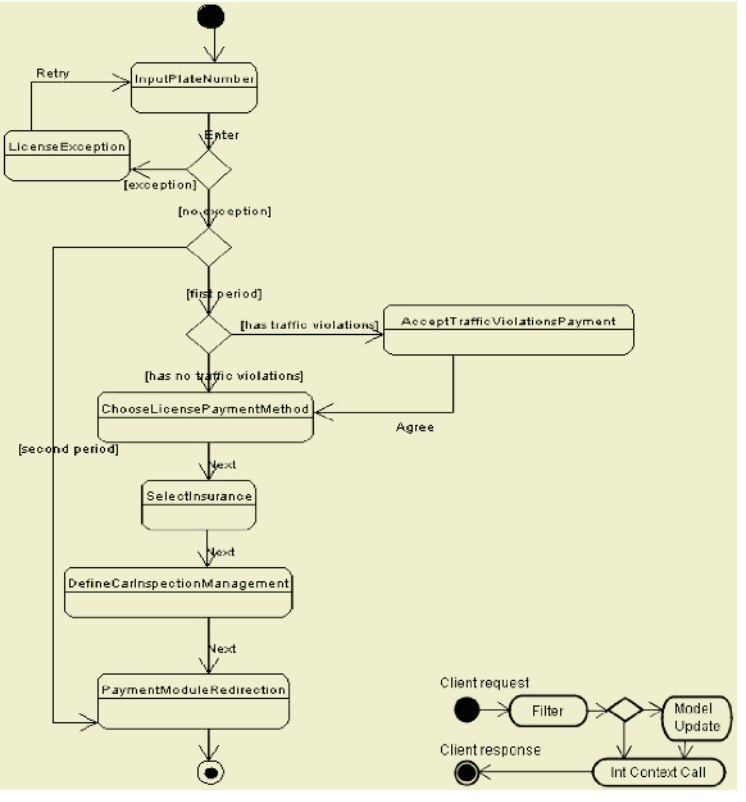

Fig. 2. Navigation Model for the example application (left). and a generic controller (right)

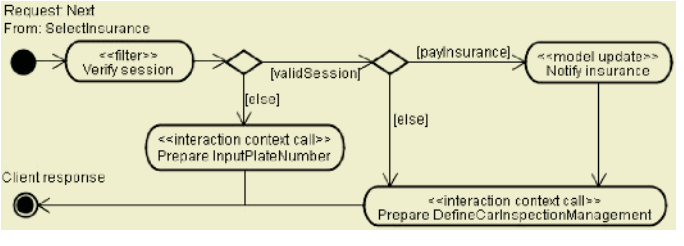

Fig. 3. A Controller Model with Filter for the example application

5 Conclusion

We have presented a methodology for creating Web applications that allows postponing platform specific details up to the very end of the project. Since the abstract artifacts relate closely to the concrete ones that appear in any MVC-based framework, the

final mapping is direct and easy to perform. In our experience, our approach can be learned by Web developers and put to use almost immediately. Future tasks include process formalization and tool development for semiautomatic mapping of abstract to concrete artifacts.

References

1. A. Ginige and S. Murugesan, "Web Engineering: An Introduction". IEEE MultiMedia, vol. 8, Nº3, 14-18 (2001).
2. J. Conallen, *Building Web Applications with UML*. The Addison-Wesley Object Technology Series. Addison Wesley, 1999.
3. J. Zhang and U. Buy, "A Framework for the Efficient Production of Web Applications", Proceedings of the Eighth IEEE International Symposium on Computers and Communication (ISCC'03) 1530-1346/03 (2003).
4. C. Kaewkasi and W. Rivepiboon, "WWM: A Practical Methodology for Web Application Modeling", Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02) 0730-3157/02 (2002).
5. L. Baresi, F. Garzotto, and P. Paolini, "Extending UML for Modeling Web Applications", Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 3, Page: 3055 (2001).
6. N. Koch and A. Kraus, "The Expressive Power of UML-based Web Engineering", Second International Workshop on Web-oriented Software Technology (IWWOST02) (2002).
7. OMG Model Driven Architecture (2004). http://www.omg.org/mda/
8. L. Ramirez and J. Navon, "Model Centric Web Development", Proceedings of the 7th Conference on Software Engineering and Applications, Marina del Rey, CA (M. Hamza editor) (2003).
9. T. Husted, *Struts in action: Building Web Applications with the Leading Java Framework*, Manning Publications Co, 2002.
10. L. Constantine, "Canonical Abstract Prototypes for Abstract Visual and Interaction Design", University of Technology, Sydney, Australia (2003).